Multi-language / multi-OS commmunication using RabbitMQ

Wil de Bruin

Who am I

- Wil de Bruin – Software / system engineer
- Wageningen, The Netherlands
- Graduated in Environmental Sciences
- Reseach microbiologist (1987)
- CTO – Founder Site4u (1994)
- CFML since Allaire ColdFusion 1.5
- Coldfusion training, consultancy, software development and hosting
- Coldbox user since 2008

Agenda

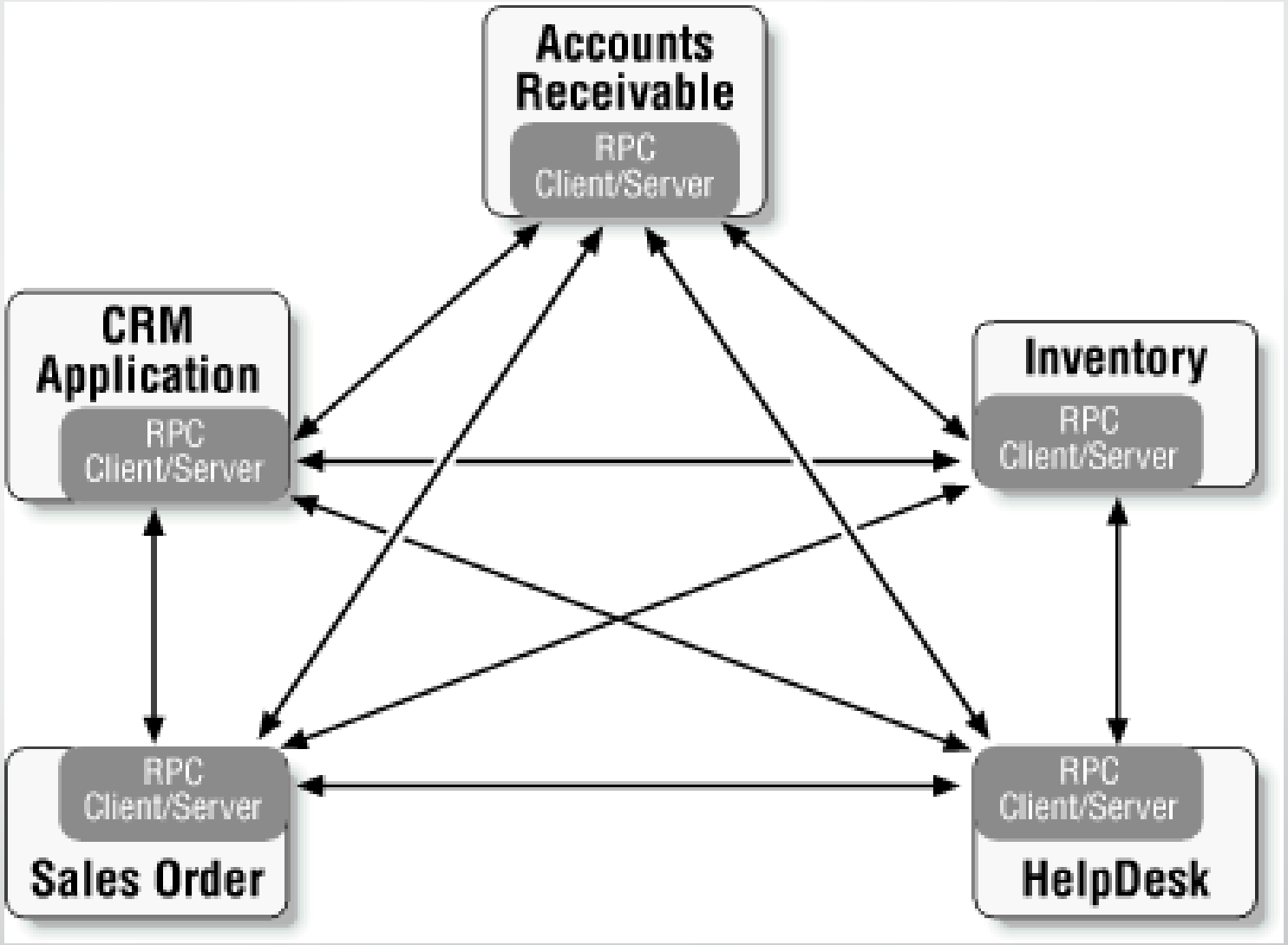
- Why RabbitMQ messaging? RPC vs Messaging
- RabbitMQ Concepts explained
- Our use case: webhosting panel
- Talking to RabbitMQ (Producers)
- Listening to RabbitMQ (Consumer)
- Conclusions

Why RabbitMQ messaging?

RPC vs Messaging

RPC

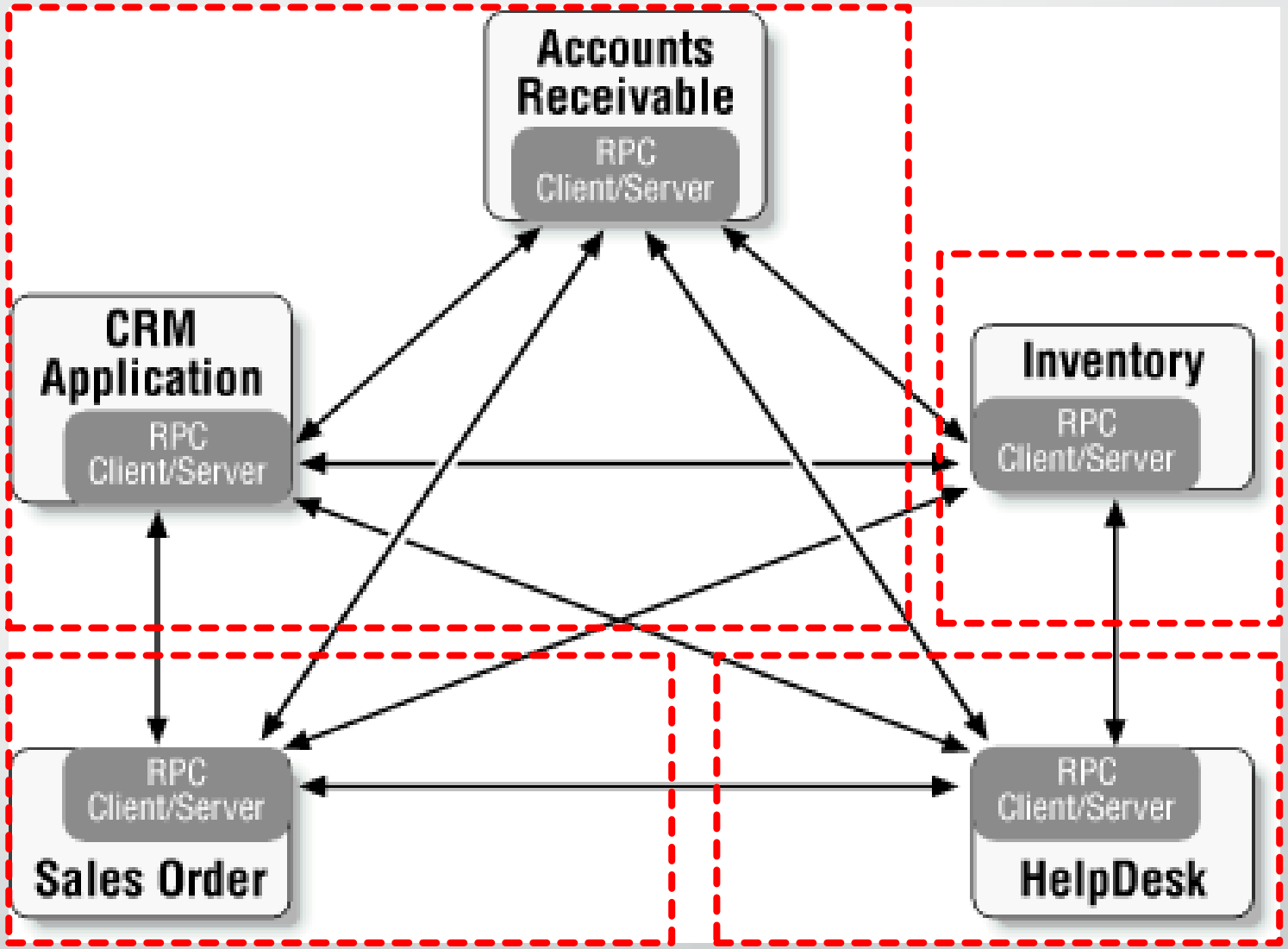
- Only one receiver
- Sender knows about receiver
- Receiver knows about sender
- Synchronous: Blocks request usage

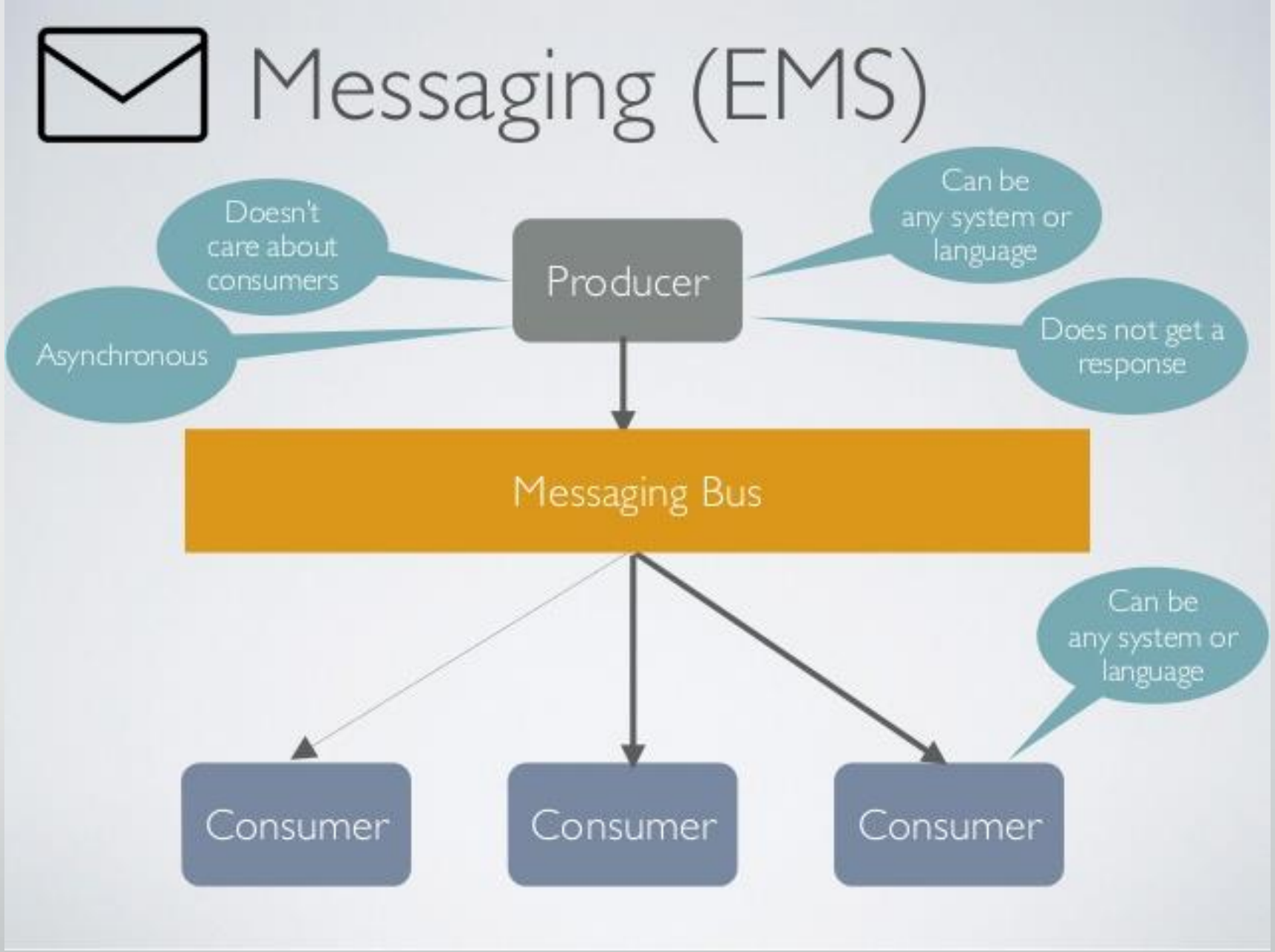


https://www.oreilly.com/library/view/java-message-service/0596000685/ch01s04.html

RPC

- Tightly coupled
- Synchronous
- Hard to configure and maintain





https://www.slideshare.net/ColdFusionConference/enterprise-messaging

Messaging advantages

- Cross platform, multiple technologies: Flexibility
- Decoupling Producers and Consumers
- Event driven programming: scalability
- Queuing
- Load balancing

Rabbit MQ concepts explained

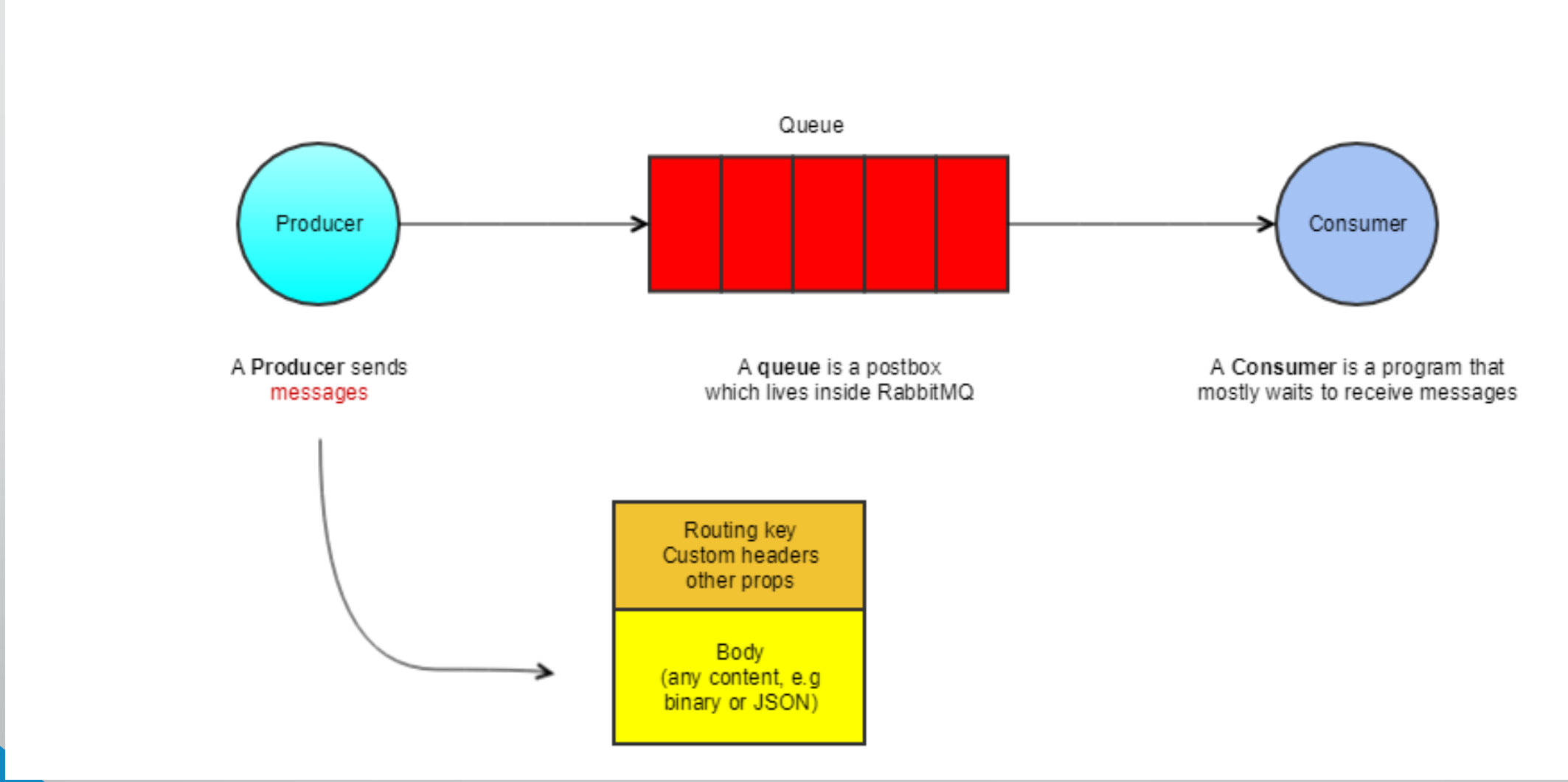
AMQP
Advanced Message Queuing Protocol (www.amqp.org)

- Binary protocol
- Diverse programming languages can communicate
- Queuing
- Routing
- Exchanges
- Secure
- Reliable

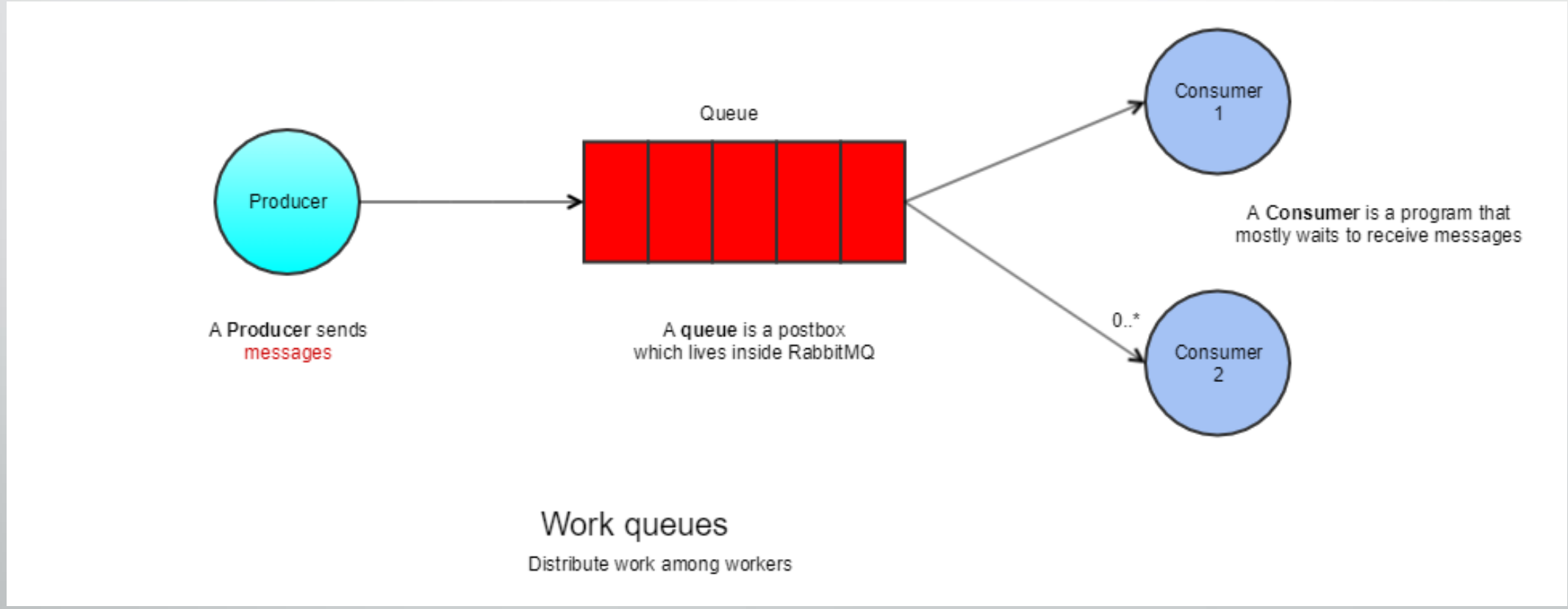
RabbitMQ

- Multiple messaging protocols (AMQP, STOMP, MQTT)
- Open source, well known
- Lots of libraries in different languages (Java, .NET, Python,Ruby, Javascript)
- Http management
- Simple concept: just accepts and forwards messages
- Developer friendly: deploy with Docker, Puppet and more

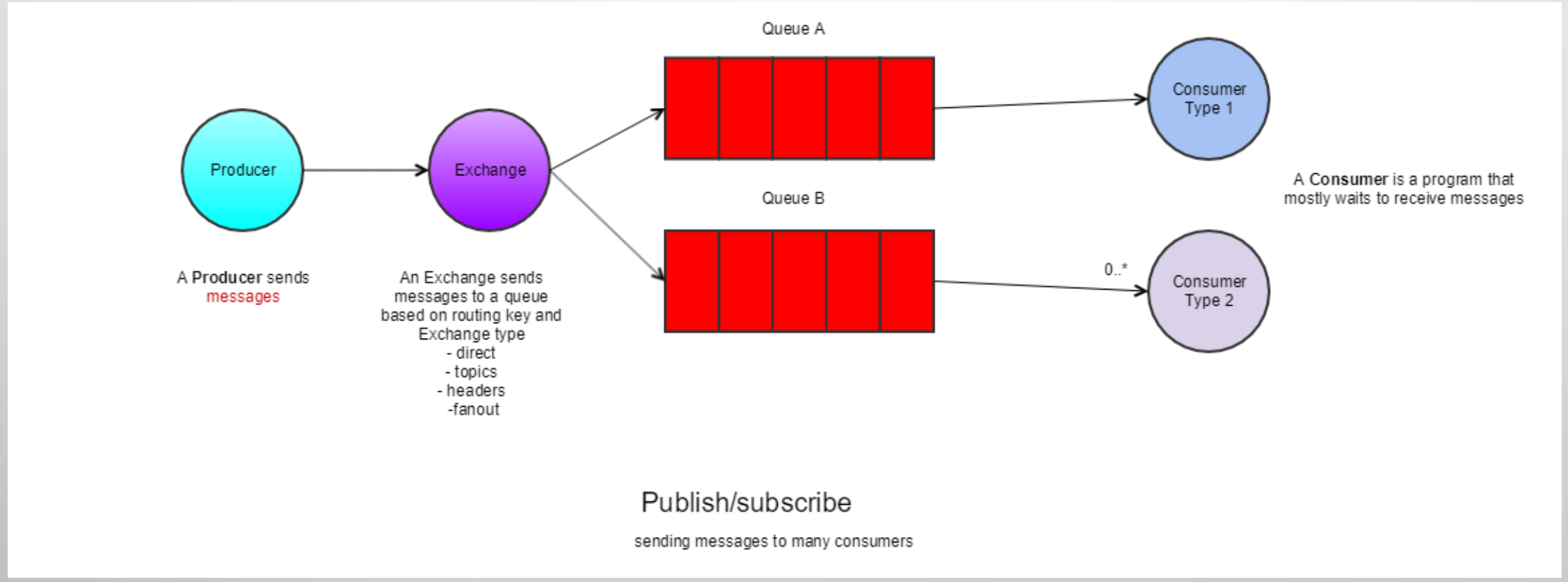
A simple message (1/5)



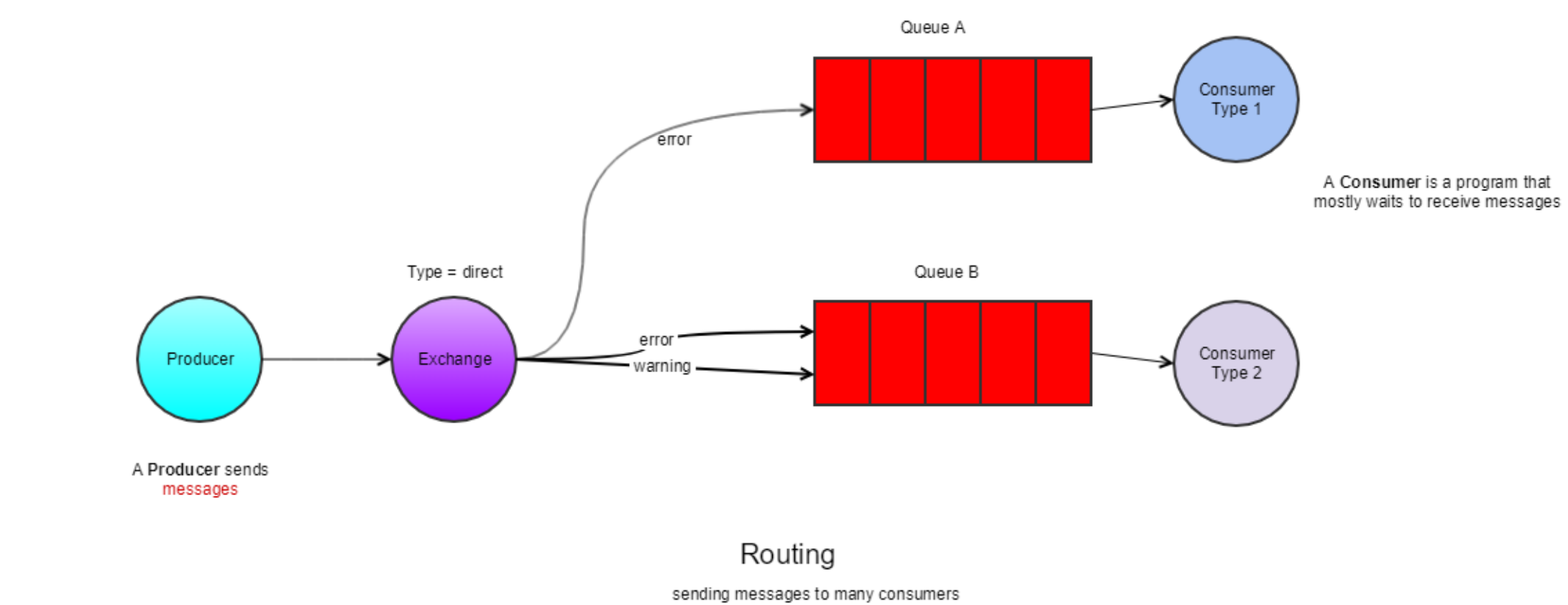
Work queue (2/5)



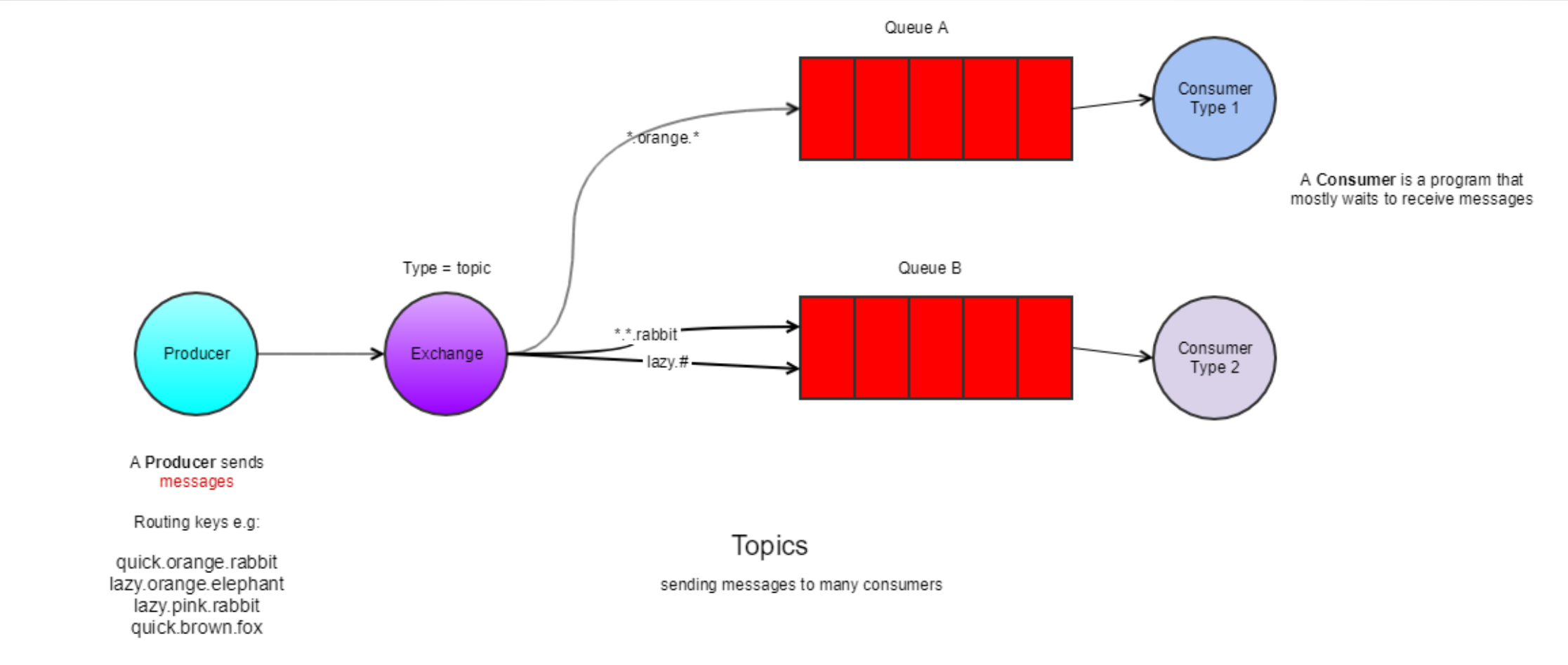
Publish/subscribe (fanout exchange) (3/5)



Routing (direct exchange) (4/5)



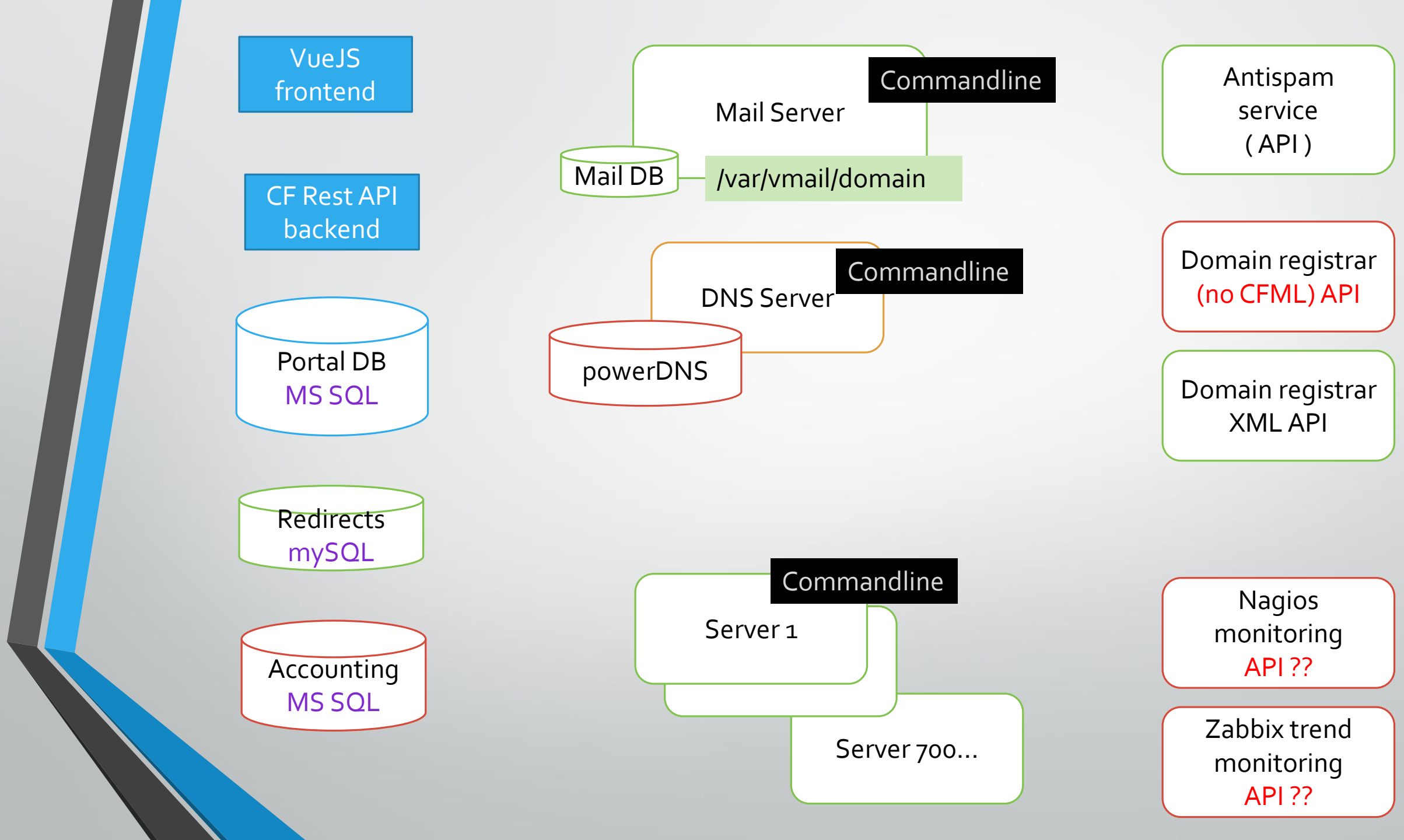
Topic exchange (5/5)



Use case:
Web hosting panel

Web hosting panel

- Dns, Email functionality: just like Cpanel, plesk etc
- Customers demanding SLA reports: integration with Monitoring
- Resource usage: VMWare integration
- Management of servers and networks: complex networks require more automation


VueJS frontend
CF Rest API backend
Portal DB MS SQL
Redirects mySQL
Accounting MS SQL
Mail Server
Commandline
Mail DB
/var/vmail/domain
DNS Server
Commandline
powerDNS
Commandline
Server 1
Server 700...
Antispam service (API)
Domain registrar (no CFML) API
Domain registrar XML API
Nagios monitoring API ??
Zabbix trend monitoring API ??

Some limitations: Mailserver

- Mail Server is OpenSource (postfix / Dovecot (IMAP) / webmail (Roundcube)
  - User config is defined in a database
  - User mailboxes as files and directories. Will NOT be deleted when removed from database.
  - Roundcube in separate database
  - Report sizing with commandline tools, no API available.
  - Environment: Linux, bash and Python

More limitations: DNS and DNSsec

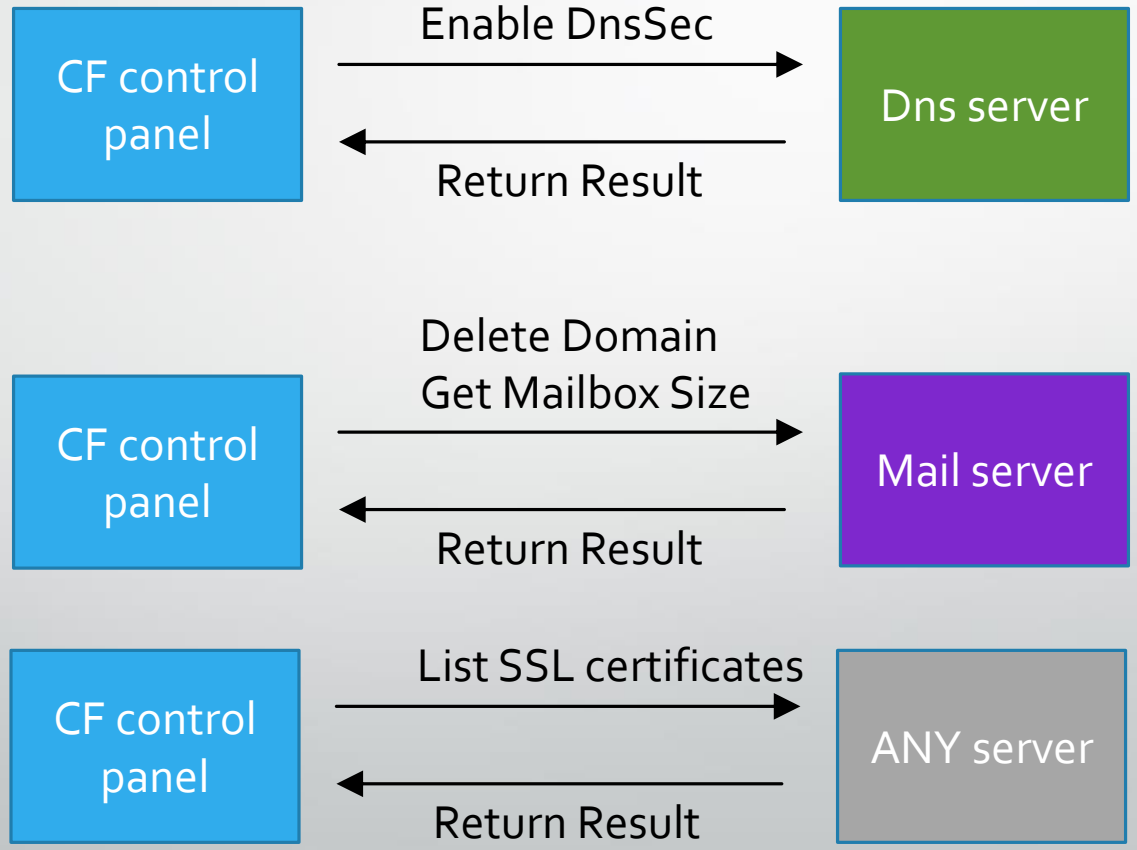
- PowerDNS has mySQL backend, but no management of DNSsec
  - Workflow
    - Create DNSsec in powerDNS (commandline)
    - Send key to registrar
      - Registrar 1: XML API, several languages
      - Registrar 2: API for CFML is hard, several other language libs available (e.g PHP)
    - Change status in control panel
  - Environment: Linux, bash, commandline,

Still more limitations: SSL and HTTP configs

- SSL installation is manual, customers need overview (commandline)
- Mixture of
  - Custom certificates from several providers
  - Let's encrypt: automated management, but how to notify our control panel?
- Environment:
  - Linux, bash, commandline, python
  - Windows, powershell

Our RabbitMQ messaging setup

Sending commands in our hosting environment



Our messaging setup

- Topic exchange
- Routing keys:
  - mysite4u.*servicename*.cmd for commands to a service
  - mysite4u.*servicename*.response for responses of a service
- Queues: **servicename**-commands and mysite4u-responses
- Exchange: mySite4u-topic
- Queues and messages are DURABLE i.e persistent.
- Special queue for unrouted messages

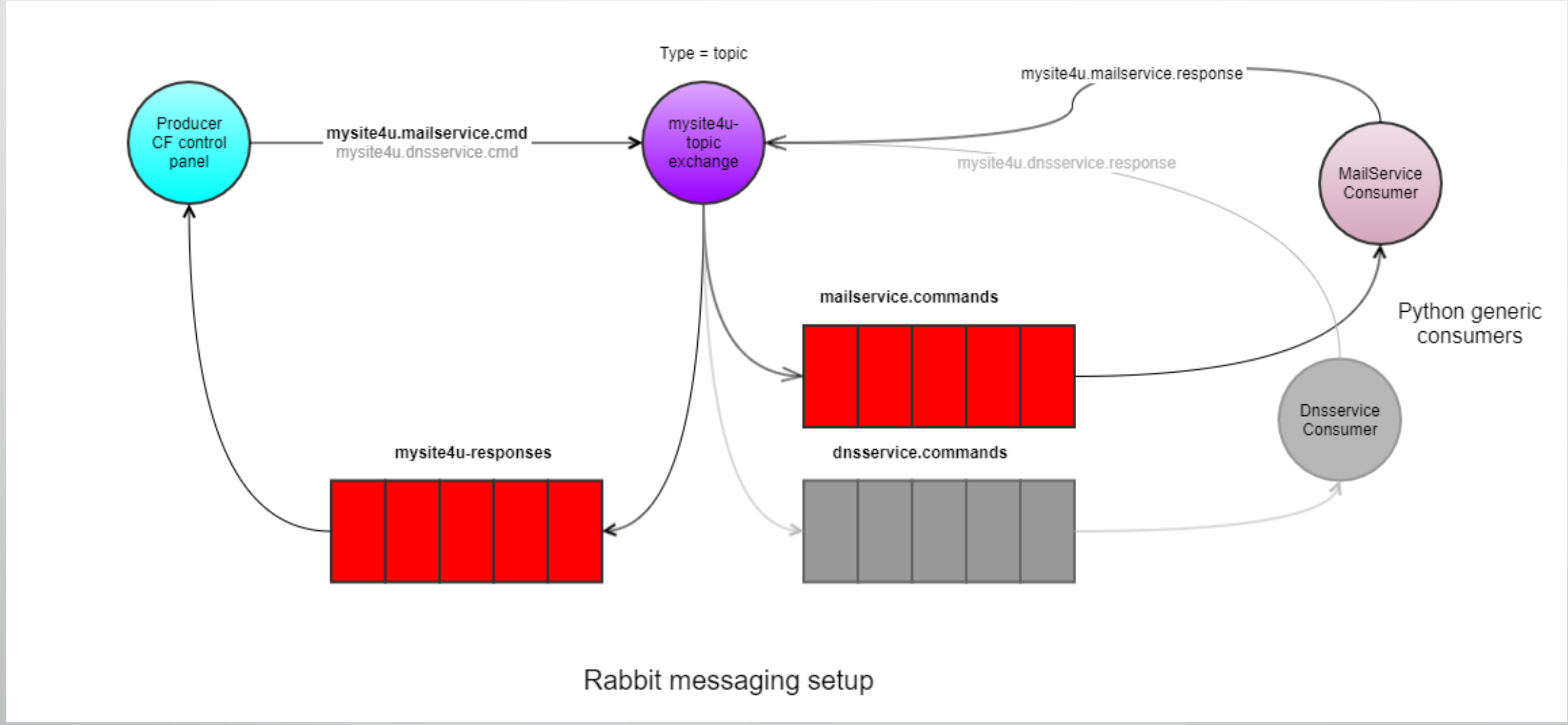
Commands and responses

COMMAND

```
Headers:
    action = string (for example: archive of getmailboxsize)


Properties:
    correlation_id: string
    content_type: string (application/json of text/plain)


Body: [string,json]
{
    "params": {
        "domainname": "example.com",
        "param2": "Second parameter"
    }
}
```

Response

```
Properties:
    correlation_id: string
    content_type: string (application/json)


Body: json
{
    data: [string, json]
    status: [ok,warning,error]
    messages: string
}
```



Rabbit messaging setup

Flexible messaging daemon for hosting servers

- RabbitMQ Consumer written in Python
- Executes any command based on configuration, e.g:

```
[mysite4u.sharedmail.cmd:archive]
script: /usr/local/scripts/archiveMail/archiveMailDomain.php $domainname
run-as: vmail
reply-to: mysite4u.sharedmail.response
```

- Output of script is piped into RabbitMQ producer response.

Talking to RabbitMQ (Producer)

Setup (1)

- Install and configure RabbitMQ
- Download https://www.rabbitmq.com/java-client.html and create javasettings in Application.cfc
  `this.javaSettings = { loadPaths = [ "lib" ], reloadOnChange = false };`
- Create mappings in wirebox.cfc

```
//mapping for connection factory
map("RabbitFactory").toJava("com.rabbitmq.client.ConnectionFactory").asSingleton();

// mapping for message properties builder
map("RabbitPropertiesBuilder").toJava("com.rabbitmq.client.AMQP$BasicProperties$Builder")
.asSingleton();
```

Setup (2)

- Make a connection onApplicationStart

```
// create connection factory
rabbitFactory.setHost( RabbitMQ.Host );
rabbitFactory.setUserName( RabbitMQ.User );
rabbitFactory.setPassword( RabbitMQ.Password );
rabbitFactory.setVirtualHost( RabbitMQ.VHost );
rabbitFactory.setHost( RabbitMQ.Host );
rabbitFactory.setPort( RabbitMQ.Port );
rabbitFactory.useSslProtocol();

// Create a shared connection for this application
application.connection = rabbitFactory.newConnection();
```

Sending a message to RabbitMQ

```
  Var DELIVERY_PERSISTENT = 2;
  var CONTENT_TYPE = "application/json";
  var Routingkey = "mysite4u." & service & ".cmd";
  try {
    var Channel = application.connection.createChannel();
    //Exchange, queue and binding are responsibilites for the Consumer
    //Create properties with property builder
    var myHeaders = { 'action' =  arguments.action };
    var props = RabbitPropertiesBuilder
      .correlationId( arguments.CorrelationId )
      .contentType( CONTENT_TYPE )
      .deliveryMode( DELIVERY_PERSISTENT )
      .headers(myHeaders)
      .build();
    var myMessage = serializeJSON( { 'params' = arguments.RabbitMessage} );
    channel.basicPublish(RabbitMQ.Exchange, Routingkey, props, createByteArray( myMessag
e ) );
    // TODO log and send to interceptor
  } catch ( any e) {
      logger.error("Error sending rabbitMQ message", e);
  } finally {
      channel.close();
   }
  }
```

How to handle returned results

- RabbitMQ RPC style ???
  https://www.rabbitmq.com/tutorials/tutorial-six-python.html
- When sending a message, use **CorrelationId**. Store CorrelationId together with type and Id of object you want to process in the result.
- When receiving a result, retrieve object info based on CorrelationId and process result

Listening to RabbitMQ (Consumer)

Consumer setup

- Read the RabbitMQ java guide: https://www.rabbitmq.com/api-guide.html
- Read some sample code https://github.com/lmajano/messaging-polyglot/tree/master/consumer
- Create a Consumer CFC. Add Logger and Interceptor properties.
- Start your Consumer onApplicationStart

Implement com.rabbitmq.client.Consumer

- add property name= "logger‟ and property name="InterceptorService";
- Sample implementation: https://github.com/lmajano/messaging-polyglot/blob/master/consumer/cfml/models/Consumer.cfc
- **handleDelivery** for the real action

```
public void function handleDelivery(consumerTag,envelope,properties,body){
  try {
    var fullmessage = {
      properties = properties,
      headers = structNew(),
      envelope = envelope,
      body = body
    }
    if ( !isNull(properties.getHeaders())){
      for (var key in properties.getHeaders()) {
        fullmessage.headers[key] = properties.getHeaders().get(key);
      }
    }
    InterceptorService.processState('onRabbitMQMessageReceived', fullmessage);
  } catch ( any e) {
    if ( logger.canError() ) {
      logger.error("#e.message# message", e);
      logger.error("failed rabbit message", fullmessage);
    }
  } finally {
    channel.basicAck( envelope.getDeliveryTag(), false );
  }
}
```

```
void function onRabbitMQMessageReceived( event, interceptData, buffer, rc, prc ){
  try {
    var correlationId = interceptData.properties.getCorrelationId();
    var decodedBody = DeSerializeJson(tostring(interceptData.body));
    var routingKey = interceptData.envelope.getRoutingKey();
    //process rabbitReply (update transaction table and find objecttype and name)
    var transReply = processRabbitReply( correlationId, decodedBody.status );
    //exit if status == error
    if ( decodedBody.status == "error" ){
      logger.error("Error in #Routingkey# for  #CorrelationId#", interceptdata);
      return;
    }
    switch(routingKey) { // action based on routing key
      case "mysite4u.sharedmail.response":
        var myDomainName=transactionReply.ObjectKeyValue;
        ResetMailTypeToNone(myDomainName);
        break;
      // … more actions;
      default:
        logger.error("invalid key #interceptData.RoutingKey#", interceptdata);
        break;;
    }
  } catch (any e) {
    logger.error("Error processing RabbitReply: #e.message#", e)
    logger.error("Error processing RabbitReply message", interceptData)
  }
}
```

Starting a CFML consumer onApplicationStart

```
    // Create new channel for this interaction
    application.channel = application.connection.createChannel();
    // Create Queue just in case, with arguments queName
    // passive, durable,exclusive, autoDelete
    application.channel.queueDeclare(RabbitMQ.EventsQueue,
      javaCast( "boolean", true ), javaCast( "boolean", false ),
      javaCast( "boolean", false ), javaCast( "null", "" )
    );
    var Consumer =  new models.Consumer( application.channel);
    Consumer.setLogger(logger);
    Consumer.setInterceptorService(InterceptorService);
    // Prepare a push consumer
    consumerTask = createDynamicProxy( Consumer,
[ "com.rabbitmq.client.Consumer" ]);
    // Consume Stream API
    param name="prc.consumertag" default="geen";
    prc.consumerTag = application.channel.basicConsume( RabbitMQ.EventsQueue,
       false, consumerTask );
```

Some CFML horror story

Writing CFML Consumers is hard….

- Lucee doesn't call onApplication stop when you stop or restart Lucee. So no way to cleanup connections.
- Lucee is caching wrong class info on **CreateDynamicProxy.** Application fails on second startup. Cleanup manually by deleting C:\Users\wbr\.CommandBox\server\6F742CC718A8B01CD51A4773E2B81B0F-mysite4u-back\lucee-5.2.9.31\WEB-INF\lucee-web\cfclasses\RPC

  Clever commandbox hack:

```
"scripts":{

"onServerStart":"rm '`server info property=serverHomeDirectory`/WEB-INF/lucee-web/cfclasses/RPC/'  --force --recurse“
}
```

- **CreateDynamicProxy** is loosing application context. So all datasources and application mappings will fail, not only in your Consumer but also in Interceptors and any other code called by your javaproxy. Unless you recreate all mappings and datasources again by adding something like this to your init

```
application action="update" datasource="myDSN"
  mappings=structOfCFMappings
```

What's missing

- Security:
  - Use SSL in rabbit connections
  - Make sure your reply is a valid one: use correlationID for command and match this in handling the reply
- Monitoring
  - Make sure your commands are processed
  - Monitor your queue lengths

Conclusions

- Messaging is very flexible, new ways of communication
- No more coding barriers, many languages can be used
- CF rabbitMQ producers easy to achieve
- CF rabbitMQ consumers are hard to configure, due to language limitations, especially push Consumers
  Pull consumers are easy but now you'll have scheduling issues
- Is CFML the best tool for a rabbit Consumer in a micro services architecture?

Thank you!

Questions?

E-mail: wil@site4u.nl
Slack: @wil-site4u